Intel® Itanium® 2 Processor

Specification Update

March 2003

Notice: The Intel® Itanium® 2 processor may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are documented in this specification update.

Document Number: 251141-009

Contents

Revision History

Date	Version	Description
March 2003	009	Added errata 53-54; added PAL version 7.40.
February 2003	008	Updated workaround for erratum 48; added erratum 52; added PAL version 7.37.
January 2003	007	Added errata 49-51; added Documentation Change 2.
December 2002	006	Added errata 47-48.
November 2002	005	Added errata 43-46; added PAL version 7.36.
October 2002	004	Added errata 38-42.
September 2002	003	Added errata 30-37; added PAL version 7.31; added Documentation Change 1; added Specification Clarification 1.
August 2002	002	Added errata 20-29.
July 2002	001	Initial release of this document.

Preface

This document is an update to the specifications contained in the Affected Documents/Related Documents table below. This document is a compilation of device and documentation errata, specification clarifications, and changes. It is intended for hardware system manufacturers and software developers of applications, operating systems, or tools.

This document may also contain information that was not previously published.

Affected Documents/Related Documents

Title	Document #
Intel® Itanium® 2 Processor at 1.0 GHz and 900 MHz Datasheet	250945
Intel® Itanium® 2 Processor Hardware Developer's Manual	251109
Intel® Itanium® Architecture Software Developer's Manual, Volume 1: Application Architecture	245317
Intel® Itanium® Architecture Software Developer's Manual, Volume 2: System Architecture	245318
Intel® Itanium® Architecture Software Developer's Manual, Volume 3: Instruction Set Reference	245319
Intel® Itanium® Architecture Software Developer's Manual Specification Update	251141
Intel® Itanium® 2 Processor Reference Manual for Software Development and Optimization	251110
Itanium® Processor Family System Abstraction Layer Specification	245359

Nomenclature

S-Spec Number is used to identify products. Products are differentiated by their unique characteristics, e.g. core speed, L3 cache size, package types, etc. Care should be taken to read all notes associated with each S-Spec number.

Errata are design defects or errors. These may cause the Intel® Itanium® 2 processor's behavior to deviate from published specifications. Hardware and software designed to be used with any given stepping must assume that all errata documented for that stepping are present on all devices.

Specification Changes are modifications to the current published specifications. These changes will be incorporated in the next release of the specifications.

Specification Clarifications describe a specification in greater detail or further highlight a specification's impact to a complex design situation. These clarifications will be incorporated in the next release of the specification.

Documentation Changes include typos, errors, or omissions from the current published specifications. These changes will be incorporated in the next release of the specifications.

Note: Errata remain in the specification update throughout the product's lifecycle, or until a particular stepping is no longer commercially available. Under these circumstances, errata removed from the specification update are archived and available upon request. Specification changes, specification clarifications, and documentation changes are removed from the specification update when the appropriate changes are made to the appropriate product specification or user documentation (datasheets, manuals, etc.).

Summary Table of Changes

The following table indicates the errata, specification changes, specification clarifications, or documentation changes which apply to the Itanium 2 processor. Intel may fix some of the errata in a future stepping of the component or in a future release of the Processor Abstraction Layer (PAL), and account for the other outstanding issues through documentation or specification changes as noted. This table uses the notations indicated below.

Codes Used in Summary Table

Stepping

X:	Errata exists in the stepping or PAL version indicated. Specification Change or Clarification that applies to this stepping.
(No mark) or (Blank box):	This erratum is fixed in listed stepping or specification change does not apply to listed stepping or PAL version.

Page

(Page):	Page location of item in this document.

Status

Doc:	Document change or update will be implemented.
Plan fix:	This erratum may be fixed in a future stepping of the component, or in a future release of PAL.
Fixed:	This erratum has been previously fixed.
NoFix:	There are no plans to fix this erratum.

Row

	Change bar to left of table row indicates this erratum is either new or modified from the previous version of this document.

Errata (Sheet 1 of 2)

No.	Processor Stepping	PAL Version					Pg.	Status	ERRATA
	B3	7.13	7.31	7.36	7.37	7.40			
1	X						13	NoFix	IA64_INST_RETIRED and IA64_TAGGED_INST_RETIRED does not count predicated off instructions
2	X						13	NoFix	Performance Monitor Interrupt raised when freeze bit is written to Performance Monitoring Counter register
3	X						13	NoFix	Priority agent requests with unit mask of I/O not counted
4	X						13	NoFix	Incorrect fault reporting on move to/from the RNAT or BSPSTORE application registers
5	X						14	NoFix	Power good deassertion affects boundary scan testing
6	X						14	NoFix	IA-32: CPUID instruction returns incorrect L3 cache size
7	X						14	NoFix	Performance Monitoring Event counters may be incorrect when using Instruction Address Range checking in fine mode
8	X						15	NoFix	Possible deadlock condition after ptc.g is issued on two-way system
9	X						15	NoFix	EPC, mov ar.pfs and br.ret instructions may combine to yield incorrect privilege level
10	X						16	NoFix	Removal of WAW hazard may lead to undefined result
11	X						16	NoFix	Unexpected data debug, data access or dirty bit fault taken after rfi instruction
12	X						17	NoFix	Incorrect privilege level may be granted if a failed speculation check precedes a privilege level change
13	X						17	NoFix	Floating-point instructions take a floating-point trap before Unimplemented Instruction Address trap
14		X					17	Fixed	PAL_MC_ERROR_INFO does not return an address for certain double bit ECC memory errors
15		X					17	Fixed	PAL_CACHE_READ and PAL_CACHE_WRITE return incorrect status for L1I cache access
16		X					18	Fixed	Unpredictable behavior if the system is awakened from low power mode by an MCA
17		X					18	Fixed	The system may lose an interrupt when SAL_CHECK reads the IVR
18		X					18	Fixed	A bus MCA nested within a recoverable or firmware-corrected bus MCA may not be handled correctly
19		X					18	Fixed	PAL reset sequence performed after a recovery check may result in incorrect system behavior
20		X					19	Fixed	PAL_HALT_LIGHT_SPECIAL provides PAL_HALT functionality
21		X					19	Fixed	PAL_TEST_PROC may access memory with the UC attribute
22	X						19	NoFix	L2 single bit data error promoted to MCA continues to flag a CMCI
23		X					19	Fixed	PAL_TEST_PROC requires specific tests be performed for correct operation
24		X					19	Fixed	PAL_TEST_INFO may return incorrect data for invalid test parameters
25		X					20	Fixed	PAL_CACHE_INIT may not function properly if levels of the cache hierarchy are specified
26		X					20	Fixed	PAL_SET_TIMEOUT may have an unexpected result when timeout = 0
27		X					20	Fixed	Concurrent MCAs that signal a BERR may not set PSP.bc correctly
28		X					20	Fixed	PAL_PLATFORM_ADDR may return an error if bit 63 is set
29		X					20	Fixed	PAL_TEST_PROC may overwrite predicate registers
30		X					20	Fixed	Recovery check fails if PAL_B is not found
31		X					21	Fixed	PAL procedure calls may have unexpected results if an incorrect PAL_B version is used
32		X					21	Fixed	Late self-test may have unexpected results during concurrent processor tests
33		X					21	Fixed	PAL_TEST_PROC may cause unexpected system behavior
34		X					21	Fixed	PAL halt procedures may overwrite predicate registers
35	X						22	NoFix	Two resets may be necessary to leave TAP test mode

Errata (Sheet 2 of 2)

No.	Processor Stepping	PAL Version					Pg.	Status	ERRATA
	B3	7.13	7.31	7.36	7.37	7.40			
36	X						22	NoFix	IA-32 instruction pointers may be overwritten under certain boundary conditions
37		X					22	Fixed	Initialization and ETM recovery may overwrite branch register
38			X				22	Fixed	PAL procedures may not save predicate register 3
39		X	X				22	Fixed	PAL_CACHE_INFO procedure may return undefined value
40			X				23	Fixed	PAL_HALT_LIGHT procedure may generate a spurious Performance Monitor Interrupt
41		X	X				23	Fixed	Unexpected system behavior after PAL_CACHE_FLUSH is executed
42		X	X				23	Fixed	PAL_TEST_PROC may not properly report self-test status
43	X						23	NoFix	PSR.ri may not reflect the correct slot upon entrance to the unimplemented address fault handler
44	X						24	NoFix	WC and WB memory attribute aliasing combine with FC and may cause processor live-lock
45	X						24	NoFix	Improper use of memory attribute aliasing may lead to out of order instruction execution
46							24	Fixed	FPSWA may not set the Denormal status flag correctly[1]
47	X						25	NoFix	Executing an rfi instruction that is located at the end of implemented physical memory can result in an unexpected unimplemented address fault
48	X						25	Fixed	IA-32: xchg instruction requires release semantics
49		X	X	X			25	Fixed	PAL MCA handler may not correctly set PSP.co bit
50		X	X	X			25	Fixed	PAL_MC_ERROR_INFO may return incorrect PSP information
51	X						26	NoFix	FPSWA trap may be missed
52	X						27	Fixed	WC evictions and semaphore operations combine to establish a potential live-lock condition
53	X						27	Fixed	The IA-32 cmpxchg8b instruction may not correctly set ZF flag
54		X	X	X	X	X	27	NoFix	PAL_TEST_PROC status return value

1. This erratum applies to FPSWA version 1.09 and earlier.

Specification Changes

No.	Processor Stepping	PAL Version					Pg.	Status	SPECIFICATION CHANGES
	B3	7.13	7.31	7.36	7.37	7.40			
									None for this revision of this Specification Update

Specification Clarifications

No.	Processor Stepping	PAL Version					Pg.	Status	SPECIFICATION CLARIFICATIONS
	B3	7.13	7.31	7.36	7.37	7.40			
1	X						29	Doc	Itanium® 2 processor behavior with CMCI to MCA Promotion enabled

Documentation Changes

No.	Processor Stepping	PAL Version					Pg.	Status	DOCUMENTATION CHANGES
	B3	7.13	7.31	7.36	7.37	7.40			
1	X						30	Doc	Itanium® 2 processor I_{CTERM} termination agent
2	X						30	Doc	Correction of pin/signal information tables

Identification Information

Intel® Itanium® 2 Package Marking

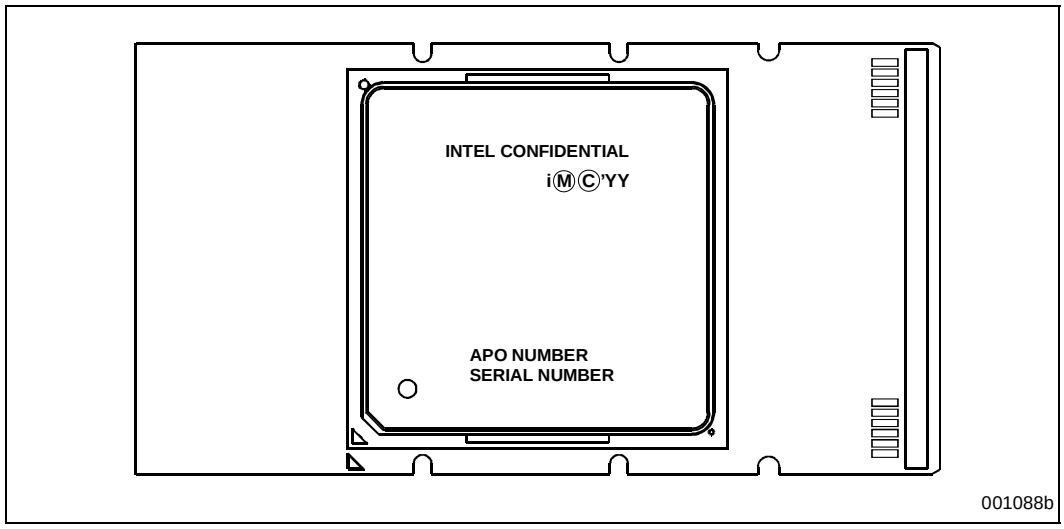
The following section details the processor top-side and bottom-side markings for the Itanium 2 processor and is provided as an identification aid. The processor top-side mark for the product is a laser marking on the Integrated Heat Spreader (IHS).

Processor Top-Side Marking

Figure 1-1 shows an example of the laser marking on the IHS. The processor top-side mark provides the following information:

- INTEL Brand/ INTEL Product
- Legal Mark
- Assembly Process Order (APO) Number
- Serial Number

Figure 1-1. Processor Top-Side Marking on IHS

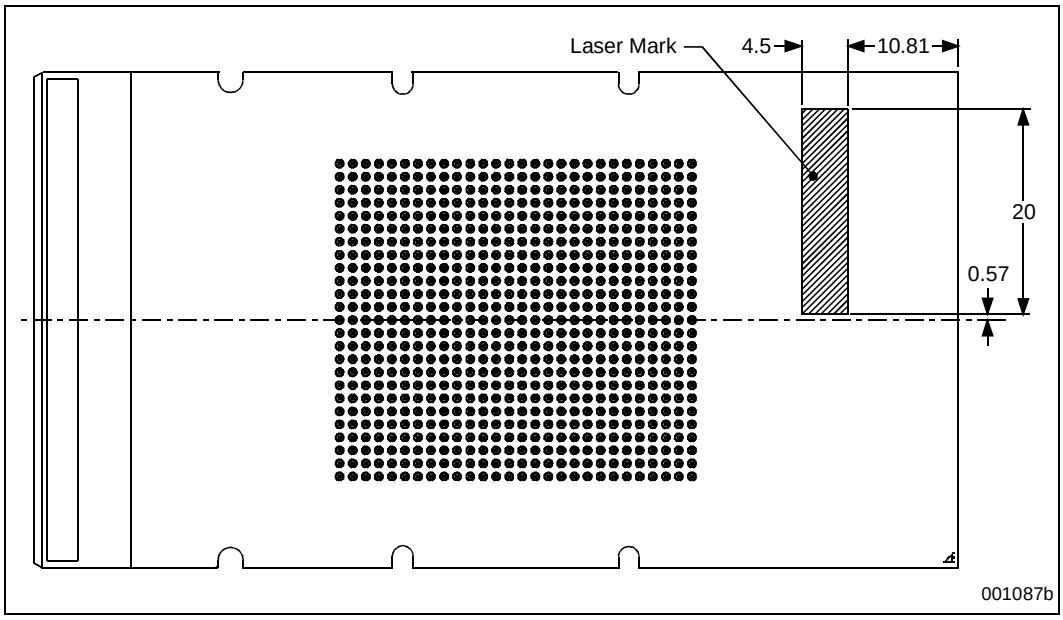


Processor Bottom-Side Marking

The processor bottom-side mark for the product is a laser marking on the pin side of the interposer. Figure 1-2 shows the placement of the laser marking on the pin side of interposer. The processor bottom-side mark provides the following information:

- Product ID
- Finish Process Order (FPO)
- Serial Number
- S-Spec
- Country of Origin

Figure 1-2. Processor Bottom-Side Marking Placement on Interposer



Intel® Itanium® 2 Processor Identification and Package Information

S-Spec Number	Processor Stepping	CPUID[1]	Speed (MHz)	L3 Size (Mbytes)
SL67U	B3	001F000704h	1000/400	1.5
SL67V	B3	001F000704h	1000/400	3
SL67W	B3	001F000704h	900/400	1.5
SL6P5	B3	001F000704h	1000/400	1.5
SL6P7	B3	001F000704h	1000/400	3
SL6P6	B3	001F000704h	900/400	1.5

1. The CPUID column in this table indicates the contents of bits 39:0 of CPUID Register 3. Bits 63:40 of this register are reserved. The Family ID for the Itanium 2 processor is 0x1F.

PAL Version	Processor Stepping	Notes
7.13	B3	
7.31	B3	
7.36	B3	
7.37	B3	
7.40	B3	

Errata

1. IA64_INST_RETIRED and IA64_TAGGED_INST_RETIRED does not count predicated off instructions

Problem: The event monitor count for instructions retired (IA64_INST_RETIRED and IA64_TAGGED_INST_RETIRED) does not include the predicated off instructions.

Implication: The IA64_INST_RETIRED/IA64_TAGGED_INST_RETIRED performance monitoring events may report an incorrect count.

Workaround: Add the PREDICATE_SQUASHED_RETIRED event monitor count to the IA64_INST_RETIRED and/or the IA64_TAGGED_INST_RETIRED event monitor count to get the intended results.

Status: For the steppings effected, see the Summary Table of Changes.

2. Performance Monitor Interrupt raised when freeze bit is written to Performance Monitoring Counter register

Problem: The Performance Monitor Freeze (PMC[0].fr) bit within the Performance Monitoring Counter (PMC) register is used to stop performance event monitoring. This can be set by software or by an event counter overflow. Due to this erratum, the processor may raise a Performance Monitor Interrupt (PMI) when the freeze bit is set by software, even when the Performance Monitor Overflow Interrupt (PMC.oi) bit is not enabled and no overflow has occurred.

Implication: The processor may generate a PMI when it's not expected to do so.

Workaround: The interrupt service routine needs to account for the spurious interrupt even if no performance monitor overflow is indicated.

Status: For the steppings effected, see the Summary Table of Changes.

3. Priority agent requests with unit mask of I/O not counted

Problem: The system bus allows for the BPRI# signal to be asserted one cycle before an ADS# is driven by the priority agent, provided no BREQ# pins are driven by the processor. Priority agent requests exhibiting this behavior are not counted by the system bus performance monitoring events when using a unit mask of 'I/O'.

Implication: The system bus performance monitoring events may report an incorrect count in this case.

Workaround: Measure the bus transactions for all bus masters (unit mask= 'ANY') and subtract from it the sum of the corresponding bus transactions on each local processor (unit mask= 'SELF').

Status: For the steppings affected, see the Summary Table of Changes.

4. Incorrect fault reporting on move to/from the RNAT or BSPSTORE application registers

Problem: Incorrect faulting behavior may be experienced under the following conditions:

1. A `mov.imm` (move immediate) to the ar.rsc register is executed in the same instruction bundle (or the next bundle with no intervening stop bits) as a mispredicted branch.
2. The mispredicted branch path includes another `mov.imm` to the same ar.rsc register, and is within two issue groups or less of the (mispredicted) branch instruction. This instruction is not executed. Also, the value moved to the rsc.mode field must be different than the value moved to rsc.mode in the `mov.imm` in step 1.

3. The correct branch path is then taken and includes a move to/from the ar.rnat or ar.bspstore registers, within the first bundle (or second bundle with no intervening stop bit) of the correct branch instruction.

Implication: When the above conditions line up (and there are no stalls or cache misses), the instruction in step 3 (move to/from ar.rnat or ar.bsp) uses the rse.mode value from the mov.imm in the mispredicted branch path instead of from instruction in step 1. As a result, there may be incorrect faulting behavior – an illegal opcode fault is missed (if rse.mode!= 0) or falsely indicated (if rse.mode = 0) and may result in inconsistent system behavior. This erratum has only been observed in a system validation environment.

Workaround: Use one of the following workarounds:

1. Use the register form of the move instruction or;
2. Ensure there is a stop bit between any `mov.imm` instruction to/from the ar.rsc registers and any subsequent branch instruction or;
3. Ensure that there is a stop bit between a “label” (branch target) and a subsequent move to/from ar.rnat/ar.bspstore.

Status: For the steppings affected, see the Summary Table of Changes.

5. Power good deassertion affects boundary scan testing

Problem: Deassertion of the PWRGOOD signal during boundary scan testing prevents the correct operation of the sampling functionality in the EXTEST and SAMPLE/PRELOAD JTAG commands.

Implication: As a result of this erratum the boundary scan chain function is disabled and will stop shifting data when the PWRGOOD signal is low.

Workaround: Keep the PWRGOOD signal asserted during boundary scan testing.

Status: For the steppings affected, see the Summary Table of Changes.

6. IA-32: CPUID instruction returns incorrect L3 cache size

Problem: The IA-32 `CPUID` instruction will always report the L3 cache size as 3 MB even for processors with a cache size of 1.5 MB.

Implication: IA-32 applications using the IA-32: `CPUID` instruction cannot rely on the cache size reported by this instruction. Native Itanium applications are not affected by this erratum and can access this information via the processor `CPUID` registers.

Workaround: Within the Linux* operating system (OS) environment, the ‘/proc/cpuinfo’ file contains this information. Within the Microsoft* OS environment this information is available through Windows API calls.

Status: For the steppings affected, see the Summary Table of Changes.

7. Performance Monitoring Event counters may be incorrect when using Instruction Address Range checking in fine mode

Problem: For Performance Monitoring Events that use Instruction Address Range Matching set to ‘Fine Mode’ (PMC: 14, bit 13 = 1), the address matching capability will be inconsistent and may yield incorrect results.

Implication: Due to this erratum the results of an event counter while using ‘Fine Mode’ may not be correct.

Workaround: Use normal mode when using Instruction Address Range checking.

Status: For the steppings affected, see the Summary Table of Changes.

8. Possible deadlock condition after ptc.g is issued on two-way system

Problem: In a two processor system, a `ptc.g` instruction is issued on processor A. The execution of the `ptc.g` on processor A, blocks the completion of a semaphore upon which processor B is waiting to become available. Concurrently processor B is issuing a long series of loads and stores with one or more instructions being retried or involves system memory access before being retired. Processor B's L2 cache entry queue, denoted as OzQ, is full and does not allow the `ptc.g` operation from processor A, entry into the L2 OzQ for completion. The `ptc.g` request will be presented again in three clock cycles. If processor B continues to execute a code sequence such that the L2 cache OzQ entries continue to be taken by other load/stores, then the `ptc.g` operation must continue to wait.

Implication: Due to this erratum, the system may deadlock while waiting for the `ptc.g` to be completed. Any break in, or completion of the code loop on processor B, including system interrupts, that allows the `ptc.g` operation to enter the L2 cache OzQ on processor B will be enough to release the deadlock condition. Additional processors will also change the time cycle necessary for this event to occur. This issue has only been observed during Random Instruction Testing in a system validation environment.

Workaround: None at this time.

Status: For the steppings affected, see the Summary Table of Changes.

9. EPC, mov ar.pfs and br.ret instructions may combine to yield incorrect privilege level

Problem: Due to certain internal timing and microarchitectural conditions, OS calls that return to user space from privilege code promote pages using a `br.ret` instruction, may not have the expected privilege level.

Using the following code sequence as an example:

```
<change of privilege level>   //epc on promote page; or br.ret
mov ar.pfs, [value];          //new pfs value has ppl < cpl
br.ret;;
```

In this case the `br.ret` is specified to not change the privilege level (`pl`) since the `br.ret` is asking to promote privilege to a numerically lower level. Current processor steppings may change current privilege level (`cpl`) to the `pl` at the beginning of the <change of privilege level>.

Implication: This erratum would result in having the `cpl` demoted and the user space application may not receive the correct privilege level. Privilege code promote page usage is limited and controlled by the OS. This issue has only been observed during random instruction testing in a system validation environment.

Workaround: Use one of the following workarounds:

1. Use an return from interrupt (`rfi`) instruction instead of `br.ret` to return from privilege code promote pages.
2. Insert a useless call-to-next bundle in all paths leading to a demoting `br.ret`.
3. PAL version 7.01 and above, have a workaround for this issue and it is enabled by default. The OS may implement one of the previous workarounds or a check mechanism, such that this PAL workaround can be disabled. Please review the PAL Release notes for details on the implementation of this workaround.

Status: For the steppings affected, see the Summary Table of Changes.

10. Removal of WAW hazard may lead to undefined result

Problem: Due to internal conditions an allowed WAW dependency may become a WAW hazard under the following circumstances:

- A move to the AR.PFS register is followed by a BR.CALL and both are executed in the same issue group, or
- A move to the AR.EC register is followed by a BR.RET and both are executed in the same issue group.

These combinations of instructions are legal WAW memory dependencies if one of the operations is predicated off. If preceding instructions (as indicated above) combine to change the predication on the BR.CALL or BR.RET from predicated true to predicated false, the processor may mistakenly decide the WAW hazard is still present and fail to recognize that the WAW has been removed which may result in an undefined value for ar.pfs or ar.ec.

The following code sequence demonstrates this issue:

```
p15 = 1;
;;
mov ar.pfs = R[x];
ld.c R[y] = [m]; //causes R[y] to be reloaded.

cmp.eq p15, p16 = R[y], R0;
(p15) br.call;
```

The RAW dependencies on `ld.c` to `cmp` and `cmp` to branch are legal. When the processor executes the issue group, the WAW hazard is present and the PFS results are undefined. If the `ld.c` misses the advanced load address table (ALAT), the `cmp` to branch will be re-executed, the new result of the `ld.c` causes the p15 value to change to false and thus eliminate the WAW. Then the processor may fail to recognize that the WAW has been removed.

Implication: An application may hang or signal an exception fault under these circumstances. The affected code sequence is not known by Intel to be generated in any current compiled code or exist in any current OS.

Workaround: Separate the predicate producing instruction from its consumer with a stop (as recommended in the *Intel® Itanium® Architecture Software Developer's Manual*, Volume 1: Application Architecture) or change the predication sequence to assure mutually exclusive predication of the instructions in the WAW dependency.

Status: For the steppings affected, see the Summary Table of Changes.

11. Unexpected data debug, data access or dirty bit fault taken after rfi instruction

Problem: A fault may be taken after a `rfi` instruction has been executed under the following circumstances. The IPSR.da or IPSR.dd bits are set to disable data debug/data access/dirty bit faults for the first Itanium processor system environment restore instruction. This is followed by an `rfi` instruction. The `rfi` instruction is followed by additional instructions that generate register stack engine (RSE) activity (`alloc, flushrs, br.ret`). The processor will see the RSE activity as valid Itanium system instructions and clear the ipsr.da/dd bits and this may result in an unexpected data debug, data access or dirty bit fault at the target of the `rfi`.

Implication: Due to this erratum an unexpected fault may be generated after an `rfi` instruction has been executed. This may slow the transition of the system into the Itanium system environment and log un-necessary errors.

Workaround: Separate the `rfi` from the RSE generating instruction by four issue groups of `nop` instructions.

Status: For the steppings affected, see the Summary Table of Changes.

12. Incorrect privilege level may be granted if a failed speculation check precedes a privilege level change

Problem: A failed speculation check instruction (`chk.s/chk.a/fchkf`) that is followed by a privilege change operation may result with the incorrect privilege level for instructions in the issue group of the privilege level change and beyond. The privilege changing instruction must occur within two clock cycles of the failed speculation check.

Implication: As a result of this erratum, the speculation check recovery code and subsequent instructions may have an incorrect privilege level.

Workaround: Do not use speculation near privilege changing instructions. The workaround for this erratum is to escalate failed speculation checks (speculation check re-steers) to the OS for recovery. This workaround is included in PAL version 7.01 and above.

Status: For the steppings affected, see the Summary Table of Changes.

13. Floating-point instructions take a floating-point trap before Unimplemented Instruction Address trap

Problem: A floating-point instruction that causes a floating-point trap and is the last instruction at the top of the physical address space should flag an Unimplemented Address trap before the floating-point trap.

Implication: The correct trap is flagged but only after the floating-point trap is taken first.

Workaround: None at this time.

Status: For the steppings affected, see the Summary Table of Changes.

14. PAL_MC_ERROR_INFO does not return an address for certain double bit ECC memory errors

Problem: PAL_MC_ERROR_INFO will report the address for the source of a double bit ECC memory error. However, under the conditions that the data with a 2xECC error was prefetched to the L2 cache and later filled into the L1 cache, the source address will not be available.

Implication: PAL_MC_ERROR_INFO will not be able to report the address of a double bit ECC error in this case. Double bit errors that are consumed in this scenario will be not be recoverable.

Workaround: None at this time.

Status: For the steppings affected, see the Summary Table of Changes.

15. PAL_CACHE_READ and PAL_CACHE_WRITE return incorrect status for L1I cache access

Problem: The PAL_CACHE_READ and PAL_CACHE_WRITE procedures should return a status value of '–7' (which indicates this operation is not supported for this *cache_type* and *level*) when attempting to read or write to/from the L1I (instruction) and L1D (data) cache. When these procedures attempt to access the L1I cache an incorrect status value will be returned.

Implication: Due to this erratum, using these PAL procedures to access the L1I cache will result in the return of an incorrect status value, implying that the L1I cache is readable/writeable by these PAL procedure calls.

Workaround: Do not use these PAL procedures to access the L1D and L1I caches.

Status: For the steppings affected, see the Summary Table of Changes.

16. Unpredictable behavior if the system is awakened from low power mode by an MCA

Problem: If the system is in low power mode and an machine check abort (MCA), BERR# or BINIT# is signaled, the PALE_CHECK handler will be called to process the error condition. However, PALE_CHECK does not disable low power mode so that it can continue execution. As soon as PALE_CHECK attempts to drain the processor queues, the system may re-enter low power mode. This may cause incomplete handling of the error event and potentially, intermittent continuation of the same event during later signaled BINIT# events.

Implication: The processor can appear to be trapped in low power mode and/or system behavior may be unpredictable.

Workaround: Do not use low power mode or call the following PAL procedures: PAL_HALT, PAL_HALT_LIGHT or PAL_HALT_LIGHT_SPECIAL.

Status: For the steppings affected, see the Summary Table of Changes.

17. The system may lose an interrupt when SAL_CHECK reads the IVR

Problem: The PAL_REGISTER_INFO procedure returns an incorrect value to indicate that reading the Interrupt Vector Register (IVR), CR65 (Configuration Register 65) has no side effects. Based on this incorrect return value, when SAL_CHECK reads the IVR while saving system state data to NVRAM, a pending interrupt may be allowed to proceed before the current process has been completed.

Implication: The SAL_CHECK procedure relies on the return values of PAL_REGISTER_INFO to know which ARs and CRs are safe to read and save off. Due to this erratum, the SAL_CHECK reads the IVR, and consequently causes the corresponding bit in the IRR to be cleared and the ISR to change. The results of the interrupt routine currently being executed may be lost.

Workaround: After calling PAL_REGISTER_INFO with *info_request* = 3, System Abstraction Layer (SAL) can force the correct return value for CR65 by setting bit 1 of *reg_info_2* to a value of one.

Status: For the steppings affected, see the Summary Table of Changes.

18. A bus MCA nested within a recoverable or firmware-corrected bus MCA may not be handled correctly

Problem: During the processing of a non-fatal bus MCA, if a second bus MCA is received the second MCA may be missed.

Implication: A bus MCA received in this scenario may be missed and result in unpredictable system behavior. If the first MCA is fatal, system behavior remains correct.

Workaround: None at this time.

Status: For the steppings affected, see the Summary Table of Changes.

19. PAL reset sequence performed after a recovery check may result in incorrect system behavior

Problem: The PAL early self-test sequence performed after a recovery check may not properly serialize outstanding memory transactions.

Implication: As a result of this erratum, memory transactions that are outstanding at the point of transition from the recovery check handler to PAL may cause a deadlock condition and possibly hang the processor.

Workaround: SAL can call the PAL_MC_DRAIN procedure before returning to PAL from recovery check to ensure that outstanding transactions have completed.

Status: For the steppings affected, see the Summary Table of Changes.

20. PAL_HALT_LIGHT_SPECIAL provides PAL_HALT functionality

Problem: The PAL_HALT_LIGHT_SPECIAL procedure does not issue the stop grant acknowledge special bus cycle.

Implication: PAL_HALT_LIGHT_SPECIAL behavior will be the same as PAL_HALT.

Workaround: None at this time.

Status: For the steppings affected, see the Summary Table of Changes.

21. PAL_TEST_PROC may access memory with the UC attribute

Problem: The ‘mem_attr’ self-test in PAL_TEST_PROC may access memory with the UC attribute, even though the ‘attributes’ parameter does not allow UC access.

Implication: PAL_TEST_PROC may access uncacheable memory that may not be supported in some systems.

Workaround: Set bit 44 of the PAL_TEST_PROC procedure self-test control word (*st_control*) to ‘1’ to skip the ‘mem_attr’ self-test.

Status: For the steppings affected, see the Summary Table of Changes.

22. L2 single bit data error promoted to MCA continues to flag a CMCI

Problem: With correctable machine check interrupt (CMCI) to MCA promotion enabled and an L2 single bit ECC data error occurs, an MCA is signaled but the CMCI continues to be raised. After the MCA is completed and the system calls the PAL_MC_RESUME procedure, a CMCI is raised if PSR.i = 1 (respond to external interrupts) and the CMCV.m = 0 (CMCI interrupts are pended).

Implication: A CMCI continues to be signaled on L2 1x ECC data errors, even if CMCI to MCA promotion is enabled.

Workaround: When enabling CMCI to MCA promotion, mask CMCIs by saving the state of CMCV.m then set CMCV.m = ‘1’. Restore the original state of CMCV.m when disabling promotion.

Status: For the steppings affected, see the Summary Table of Changes.

23. PAL_TEST_PROC requires specific tests be performed for correct operation

Problem: PAL_TEST_PROC self-test requires three specific tests be performed, otherwise the PAL procedure may report false failures or unexpected behavior.

Implication: The PAL_TEST_PROC procedure must perform the virtual hash page table (VHPT) test (bit 34), late floating-point test (bit 41) and RSE test (bit 45). Otherwise the system may have unexpected behavior or false test failures may be indicated.

Workaround: Bits 34, 41 and 45 in the PAL_TEST_PROC self-test control word (*st_control*) should be left at the default settings of ‘0’ so these tests are performed.

Status: For the steppings affected, see the Summary Table of Changes.

24. PAL_TEST_INFO may return incorrect data for invalid test parameters

Problem: The PAL_TEST_INFO procedure may return incorrect data or status if the input arguments are not valid or are out of range for a given parameter.

Implication: Calling the PAL_TEST_PROC procedure with invalid inputs may result in incorrect data and/or status instead of indicating invalid arguments.

Workaround: Ensure that PAL_TEST_INFO input parameters are valid and within the argument’s range.

Status: For the steppings affected, see the Summary Table of Changes.

25. PAL_CACHE_INIT may not function properly if levels of the cache hierarchy are specified

Problem: PAL_CACHE_INIT does not function properly when caches are selected individually.

Implication: A call to initialize the L1D cache may hang the processor and a call to initialize any other cache structure may fail and return an error.

Workaround: Call the PAL_CACHE_INIT procedure with level = –1 to initialize all caches.

Status: For the steppings affected, see the Summary Table of Changes.

26. PAL_SET_TIMEOUT may have an unexpected result when timeout = 0

Problem: Setting the input parameter timeout = 0 will disable the processor watchdog timer feature.

Implication: Calling PAL_SET_TIMEOUT with timeout = 0 disables the internal processor timeout function.

Workaround: Do not set the timeout parameter to ‘0’.

Status: For the steppings affected, see the Summary Table of Changes.

27. Concurrent MCAs that signal a BERR may not set PSP.bc correctly

Problem: In the case of concurrent MCAs that should result in BERR assertion, the PALE_CHECK handler may not set the PSP.bc (bus check error) bit before handing off to SAL.

Implication: As a result of this erratum, PAL_MC_ERROR_INFO will indicate that a bus error occurred, but the PSP at hand-off to SAL_CHECK will not.

Workaround: None at this time.

Status: For the steppings affected, see the Summary Table of Changes.

28. PAL_PLATFORM_ADDR may return an error if bit 63 is set

Problem: PAL_PLATFORM_ADDR should ignore bit 63 of the *address* argument. If this PAL procedure is called with bit 63 set to ‘1’ in the *address* argument, the procedure incorrectly returns status = –2 (invalid argument).

Implication: Due to this erratum, calling PAL_PLATFORM_ADDR with bit 63 of the address set to ‘1’ will return a status of ‘invalid argument’.

Workaround: Bit 63 should be set to ‘0’ when calling the PAL_PLATFORM_ADDR procedure to avoid this issue.

Status: For the steppings affected, see the Summary Table of Changes.

29. PAL_TEST_PROC may overwrite predicate registers

Problem: PAL_TEST_PROC may overwrite predicate registers pr4 and pr5, which should be preserved by the procedure.

Implication: PAL_TEST_PROC may modify pr4 or pr5, resulting in undefined behavior.

Workaround: Code calling this PAL procedure can save and restore these predicate registers around the PAL_TEST_PROC procedure.

Status: For the steppings affected, see the Summary Table of Changes.

30. Recovery check fails if PAL_B is not found

Problem: SAL may not be able to complete a recovery check when no PAL_B is present. The I/O port address, interrupt block and other features may not be available for SAL when recovery check is entered from PAL_A_SPEC.

Implication: Recovery check may fail if PAL_B is not available or is invalid.

Workaround: Ensure that the firmware interface table (FIT) entry for PAL_B points to a valid and correct version of PAL_B.

Status: For the steppings affected, see the Summary Table of Changes.

31. PAL procedure calls may have unexpected results if an incorrect PAL_B version is used

Problem: PAL procedures that call PAL_B may not provide the expected results if the first PAL_B entry in the FIT points to an incorrect version of PAL_B.

Implication: PAL procedures may fail if the PAL_B entry in the FIT is for an incorrect version.

Workaround: Ensure that the FIT entry for PAL_B points to the correct version.

Status: For the steppings affected, see the Summary Table of Changes.

32. Late self-test may have unexpected results during concurrent processor tests

Problem: While running PAL_TEST_PROC concurrently on more than one processor and the processors happen to access the same memory address space, a snoop may cause the ALAT test to fail.

Implication: If a processor self-test procedure is using the same memory space for concurrent processor testing, the ALAT test may fail and cause one processor to enter a spin loop.

Workaround: The ALAT test can be bypassed by setting bit 46 of the PAL_TEST_PROC self-test control word to '1'.

Status: For the steppings affected, see the Summary Table of Changes.

33. PAL_TEST_PROC may cause unexpected system behavior

Problem: The PAL_TEST_PROC 'late floating-point load/store test' may overwrite the fr2-fr5 and fr30-fr31 floating-point registers and the Bank 0 gr16–gr23 general registers may be overwritten by the ALAT, VHPT, translation lookaside buffer (TLB) and memory attributes tests.

Implication: PAL_TEST_PROC may corrupt the following registers: Bank 0 gr16–gr23 (general registers) and the fr2– fr5, fr30–fr31 (floating-point registers).

Workaround: Use different registers or save/restore the contents before/after running PAL_TEST_PROC.

Using the self-test control word of the PAL_TEST_PROC procedure, set the following bits to '1': To avoid corrupting the Bank 0 general registers do not run the ALAT (bit 46), VHPT (bit 35), TLB (bit 33) and mem_attr (bit 44) tests. To avoid corrupting the floating-point registers do not run the late_fp_ld_st (bit 40) test.

Status: For the steppings affected, see the Summary Table of Changes.

34. PAL halt procedures may overwrite predicate registers

Problem: Predicate registers pr1, pr2 and pr3 may be overwritten by the PAL_HALT, PAL_HALT_LIGHT and PAL_HALT_LIGHT_SPECIAL procedures.

Implication: As a result of this erratum, pr1, pr2 and p3 may be overwritten.

Workaround: Save and restore the predicate registers, as needed when calling these PAL procedures.

Status: For the steppings affected, see the Summary Table of Changes.

35. Two resets may be necessary to leave TAP test mode

Problem: After accessing the test access port (TAP), issuing a RESET# may result in the processor entering an idle state instead of beginning normal operation. Signaling a second RESET# may be necessary to properly reinitialize the system under these conditions.

Implication: Due to this erratum, a second RESET# may be required to properly reinitialize the processor after the TAP port has been accessed. Normal system operation and boot process is not affected.

Workaround: Issue two resets to properly reinitialize the processor after accessing the TAP port.

Status: For the steppings affected, see the Summary Table of Changes.

36. IA-32 instruction pointers may be overwritten under certain boundary conditions

Problem: Under certain internal conditions involving branch prediction and multiple branch instructions, IA-32 instruction pointers may be overwritten and result in IA-32 instructions being executed out of order or incorrectly. An affected code sequence would have consecutive branch instructions that have started execution before being cancelled.

Implication: Due to this erratum, IA-32 instruction pointers may be over-written resulting in incorrect IA-32 instruction execution.

Workaround: A workaround for this erratum is included in PAL version 7.31.

Status: For the steppings affected, see the Summary Table of Changes.

37. Initialization and ETM recovery may overwrite branch register

Problem: PAL INIT recovery code may overwrite br0, when it saves the system environment to the min-state save area. This erratum also affects the recovery path of an enhanced thermal management (ETM) alert that is generated while a system is in a low power mode.

Implication: INIT and ETM recovery code may overwrite br0, which prevents recovery with PAL_MC_RESUME and may result in unexpected system behavior.

Workaround: PAL version 7.31 fixes this issue.

Status: For the steppings affected, see the Summary Table of Changes.

38. PAL procedures may not save predicate register 3

Problem: The following PAL procedures may not properly save and restore predicate register pr3. The affected PAL procedures are:

PAL_CACHE_INIT, PAL_CACHE_LINE_INIT, PAL_CACHE_READ, PAL_CACHE_WRITE, PAL_CAR_INIT, PAL_COPY_INFO, PAL_COPY_PAL, PAL_PROC_SET_FEATURES, PAL_TEST_PROC

Implication: Predicate register 3 may be overwritten by the PAL procedures listed above.

Workaround: Save and restore pr3, as needed, when calling the aforementioned PAL procedures.

Status: For the steppings affected, see the Summary Table of Changes.

39. PAL_CACHE_INFO procedure may return undefined value

Problem: The PAL_CACHE_INFO procedure could return an invalid value in the config_info_1 '*at*' (cache memory attributes) field. When requesting information for the L2 and L3 cache, the '*at*' field may contain the value of 2, which is undefined.

Implication: The PAL_CACHE_INFO procedure, *cache memory attributes* field may return an undefined value.

Workaround: None at this time.

Status: For the steppings affected, see the Summary Table of Changes.

40. PAL_HALT_LIGHT procedure may generate a spurious Performance Monitor Interrupt

Problem: The PAL_HALT_LIGHT procedure may not properly set the value of the PMV.m bit on return from a low power state and as a result, a spurious PMI may be generated.

Implication: A spurious PMI may be indicated when using the PAL_HALT_LIGHT procedure.

Workaround: Set the PMV.m bit to '1' (to mask PMIs) before calling PAL_HALT_LIGHT. Set the PMV.m bit to '0' on return from the PAL_HALT_LIGHT procedure.

Status: For the steppings affected, see the Summary Table of Changes.

41. Unexpected system behavior after PAL_CACHE_FLUSH is executed

Problem: The PSR.ic bit is not restored after the PAL_CACHE_FLUSH procedure is executed with cache_type = 2. This may result in unexpected behavior when an interrupt is received after calling PAL_CACHE_FLUSH.

Implication: The system may not respond to interrupts as expected after PAL_CACHE_FLUSH is executed with cache_type = 2.

Workaround: Save and restore the PSR.ic bit as necessary, before and after calling the PAL_CACHE_FLUSH procedure.

Status: For the steppings affected, see the Summary Table of Changes.

42. PAL_TEST_PROC may not properly report self-test status

Problem: In the case that some PAL_TEST_PROC self-test functions fail, the *test_status* field may not indicate which self-test function has failed. Instead the failed test function may be raised as an initialization failure and the procedure will enter an infinite loop.

Implication: The PAL_TEST_PROC procedure may enter an infinite loop as a result of some failed self-tests, instead of operating in a functionally restricted manner.

Workaround: None at this time.

Status: For the steppings affected, see the Summary Table of Changes.

43. PSR.ri may not reflect the correct slot upon entrance to the unimplemented address fault handler

Problem: In the case of an `rfi` instruction that targets an instruction in slot 1 or 2 and the interrupt instruction pointer (IIP) points to an unimplemented physical address, the PSR.ri may point to slot 0 instead of slot 1 or 2 as expected. The required conditions to expose this erratum are: The processor is in physical address mode (PSR.it=0) and the IIP points to a physical memory address that is unimplemented.

Implication: When the processor attempts to execute on the indicated instruction bundle an unimplemented address fault will be taken and the restart instruction will indicate slot 0. Since no instruction in slot 0, 1, or 2 is executable under these conditions, there is no useful information lost when the unimplemented address fault is taken.

Workaround: None at this time.

Status: For the steppings affected, see the Summary Table of Changes.

44. WC and WB memory attribute aliasing combine with FC and may cause processor live-lock

Problem: Under certain conditions involving write coalescing (WC) stores and the execution of a flush cache (`fc`) instruction, the `fc` may not be able to proceed until the WC buffers have been emptied, resulting in a live-lock condition.

The live-lock is armed when one or more WC stores (st [A]) occur and allocate space in the processor's WC buffer. A store or load (st/ld [B]) with a writeback (WB) memory attribute is issued followed immediately by an `fc` (`fc[C]`) instruction. The `fc` is targeted to a virtual address with the same physical address as address [A], but with a WB memory attribute instead of WC. If address [B] shares the same physical address bits 14:7 with the flush cache target address [C], then the processor may live-lock.

Implication: This memory attribute aliasing (MAA) scenario is likely to occur for a short time in OS code page tear down or where a code page was previously accessed with the WC attribute, but is now implicitly considered to have WB attributes because memory translation has been disabled (PSR.dt=0).

Documented in the *Intel® Itanium® Architecture Software Developer's Manual*, Volume 2, Section 4.4.11, as part of the process to properly transition to a new memory attribute, an `fc` instruction should be issued to flush the WC buffers. However, the text also states that a memory fence (`mf`) instruction should precede the `fc` instruction. Properly following this transition procedure will be sufficient to avoid the live-lock condition.

Workaround: Precede `fc` instructions with `mf` instructions where WC buffers may be non-empty.

Status: For the steppings affected, see the Summary Table of Changes.

45. Improper use of memory attribute aliasing may lead to out of order instruction execution

Problem: An `fc` instruction is issued to a virtual memory address that has been aliased as uncacheable (UC). This is immediately followed by a load/store to a WB memory address that points to same physical memory address that is targeted by the `fc`. Due to internal conditions, the load/store may be filled from the L2 cache rather then being filled from memory after the `fc` has been completed.

Implication: Using MAA in this manner requires the proper transitioning sequence as noted in the *Intel® Itanium® Architecture Software Developer's Manual*, Volume 2, Section 4.4.11. Under these conditions, the order of operations observed directly on the system bus (by using a logic analyzer for example) may appear to be out of order, however there is no functional impact because the result of instruction execution will always be correct internally.

Workaround: None at this time.

Status: For the steppings affected, see the Summary Table of Changes.

46. FPSWA may not set the Denormal status flag correctly

Problem: In some cases when the Floating-Point Software Assistant (FPSWA) handles the following floating-point operation using the specified floating-point class/subclass types, the FPSWA may not return the correct Denormal/Unnormal (D) status flag setting in the Floating-Point Status Register (FPSR.sf0:8).

The affected operation is: Infinity * unnormalized number - Infinity = QNaN Indefinite

Implication: As a result of this erratum, the FPSWA may indicate a Denormal/Unnormal exception fault where none has occurred.

Workaround: The FPSWA version 1.12 fixes this issue.

Status: For the steppings affected, see the Summary Table of Changes.

47. Executing an rfi instruction that is located at the end of implemented physical memory can result in an unexpected unimplemented address fault

Problem: Due to this erratum, when the processor is in physical mode and an `rfi` instruction at the end of physically implemented memory is executed, the processor will take an unimplemented address fault regardless of the real target of the `rfi` (IIP).

Implication: On a platform that supports the full 50 bits of physical address, under the above conditions an unexpected unimplemented address (UIA) fault could occur and the result depends upon the implementation of the UIA fault handler. This issue has only been observed in a pre-silicon simulation environment.

Workaround: Do not place an `rfi` instruction at the end of implemented physical memory.

Status: For the steppings affected, see the Summary Table of Changes.

48. IA-32: xchg instruction requires release semantics

Problem: The IA-32: `xchg` instruction can execute and write a value without it being explicitly ordered with respect to other IA-32 stores. The IA-32 memory model is strongly ordered and requires loads to have acquire (.acq) semantics and stores to have release (.rel) semantics to be executed in proper order. As a result of this requirement the `xchg` instruction requires the use of .acq and .rel semantics but only provides .acq semantics.

Implication: Due to this erratum, store operations may not be committed to memory in order with respect to IA-32 `xchg` operations.

Workaround: None at this time. PAL version 7.37 includes a fix for this issue.

Status: For the steppings affected, see the Summary Table of Changes.

49. PAL MCA handler may not correctly set PSP.co bit

Problem: The PAL MCA handler may not set the continuable bit (PSP.co) for potentially recoverable errors.

Implication: If the PSP.co bit is not set on recoverable errors, the OS and/or application may terminate when they could have potentially recovered from the error.

Workaround: None at this time.

Status: For the steppings affected, see the Summary Table of Changes.

50. PAL_MC_ERROR_INFO may return incorrect PSP information

Problem: When the PAL MCA handler has detected a fatal condition or has requested a SAL_MC_RENDEZ procedure call, the PSP returned from the PAL_MC_ERROR_INFO procedure may not contain all error information.

Implication: If SAL_CHECK is using the PSP returned from the PAL_MC_ERROR_INFO procedure call, some error information maybe missing which could result in application termination or a system hang.

Workaround: SAL_CHECK should use the PSP data at PALE_CHECK hand off rather than from PAL_MC_ERROR_INFO.

Status: For the steppings affected, see the Summary Table of Changes.

51. FPSWA trap may be missed

Problem: For Itanium 2 processor floating-point operations, when a *tiny*[1] result is computed (this usually corresponds to an underflow occuring), the processor should defer the computation to the FPSWA handler. In most cases, FPSWA will convert the result to a denormalized value that can be represented within the specified precision. However, for an extremely limited set of conditions, the processor fails to recognize this underflow and does not take the appropriate FPSWA trap.

Implication: Exposure to this issue occurs only under the following conditions:

1. Execution of one of the following instructions: `fma`, `fms`, `fnma`, `fpma`, `fpms`, `fpnma`.
2. The input operands for `fma`, `fms`, and `fnma` instructions (with or without.s or.d completers) must be capable of containing any combination of 64 bits in their significand, in register format. (If the significands of the operands are limited to less than 64 bits, the operation is not affected.)
3. The computed result is precisely $\pm 1.0 \times 2^{(Emin-1)}$ [2]. This is a necessary (but not sufficient) condition as only an extremely small subset of the possible input operand combinations that generate a result of $\pm 1.0 \times 2^{(Emin-1)}$ actually lead to a missed FPSWA trap. There must be a massive and specific cancellation generating the result prior to rounding to the destination precision.

For operations meeting these conditions, a small subset will not take the FPSWA trap. In these cases, the result ($\pm 1.0 \times 2^{(Emin-1)}$) will not be representable within the floating-point format specified. For example, assuming single precision mode, the result would be $\pm 1.0 \times 2^{-127}$. Normally, the FPSWA handler converts this result to a denormalized value in the form of $\pm 0.1 \times 2^{-126}$ to fit within the single precision exponent format. Without this conversion the following impacts may be observed:

- For `fma`, `fms`, and `fnma` operations (with or without.s or.d completers) with FPSR.wre=0 [3], the result in the register file is numerically correct and may be used for subsequent floating-point operations without issue. However, storing this value to memory (using stfs, stfd or stfe as appropriate) will result in a correctly signed zero instead of $\pm 0.1 \times 2^{Emin}$. This is equivalent to what occurs for the "Flush-To-Zero" (FTZ)[4] mode of operation.

  It is possible to preserve the correct numerical result (i.e. 1.0×2^{-127} for the single precision example above) by using the `stf.spill` instruction for stores and the `ldf.fill` instruction for any subsequent loads.
- For register precision `fma`, `fms`, and `fnma` operations (with or without.s or.d completers) with FPSR.wre=1, the result should be $\pm 1.0 \times 2^{-65535}$. However, the result in the register file will be $\pm 1.0 \times 2^{-16382}$ in the form of a double-extended precision value.
- For parallel floating-point instructions (`fpma`, `fpms`, and `fpnma`), the result is stored in the register file as a correctly signed zero instead of $\pm 1.0 \times 2^{(Emin-1)}$. Parallel floating-point instructions are not used in any known compiled code.

Workaround: For the vast majority of floating-point usage models, no workaround is recommended. The issue is limited to an extremely small subset of possible floating-point operations with a typical impact of replacing a tiny value ($\pm 1.0 \times 2^{(Emin-1)}$) with a correctly signed zero. Any error due to this issue is typically less, in absolute value, than the majority of rounding errors that normally occur for floating-point operations. For applications requiring a workaround, the following actions are required:

---

1. A result is defined as tiny if it lies between -2^{Emin} and $+2^{Emin}$ after rounding to the destination precision with unbounded exponent range. Reference the *Intel® Itanium® Architecture Software Developer's Manual* or IEEE Standard 754-1985 for Binary Floating-Point Arithmetic for any additional clarifications.
2. For single precision, Emin = –126; for double precision, Emin = –1022; for double-extended precision, Emin = –16382; for register format, Emin = –65534.
3. Reference the *Intel® Itanium® Architecture Software Developer's Manual* for Floating-point Status Register (FPSR) bit definitions.
4. FTZ mode causes tiny results to be truncated to the correctly signed zero.

1. For `fma`, `fms`, and `fnma` operations (with or without.s or.d completers) with FPSR.wre=0, avoid input operands with 64-bit significands or use the `stf.spill` instruction for stores and the `ldf.fill` instruction for any subsequent loads.
2. Do not use register precision (FPSR.wre=1) for `fma`, `fms`, and `fnma` operations.
3. Do not use parallel floating-point operations (`fpma`, `fpms`, and `fpnma`).

Status: For the steppings affected, see the Summary Table of Changes.

52. WC evictions and semaphore operations combine to establish a potential live-lock condition

Problem: In the case that multiple processors are sharing memory space; when stores to WC memory are closely followed by semaphore operations to cacheable memory, the semaphore operations may block forward progress of the WC evictions. The semaphore will not be able to proceed until the WC stores are completed. As a result a live-lock condition is established between the WC evictions and the semaphore.

Implication: If the live-lock conditions are maintained, the system will eventually signal a BINIT. Other system activity or external interrupts may change availability of the system bus allowing the live-lock condition to be broken and the system will proceed as normal.

Workaround: None at this time. PAL version 7.37 includes a fix for this issue.

Status: For the steppings affected, see the Summary Table of Changes.

53. The IA-32 cmpxchg8b instruction may not correctly set ZF flag

Problem: The IA-32 `cmpxchg8b` instruction should set the Zero Flag (ZF) flag to 1 and update memory when the compare operation is successful. However, if due to memory contention, the upper four bytes (bits 63:32) of the targeted memory are changed during execution of the instruction and the lower four bytes remain unchanged, the ZF flag may be incorrectly set to 1, even though the upper four bytes of the compare are not equal.

Implication: If this erratum occurs, two processors in a multiprocessor environment can end up owning the same memory locations when there should be autonomous ownership.

The failing scenario can only occur in a multiprocessor system where there is heavy contention for the targeted memory location. It also requires that another processor manages to update only the upper four bytes of the targeted memory location during a very small timing window just prior to execution of the compare.

This erratum only affects the `cmpxchg8b` form of the IA-32 `cmpxchg` instruction and has only been observed in a synthetic test environment.

Workaround: PAL version 7.40 includes a fix for this erratum.

Status: For the steppings affected, see the Summary Table of Changes.

54. PAL_TEST_PROC *status* return value

Problem: The PAL_TEST_PROC procedure returns *status* = –3 when the call has completed successfully and some self-test errors have occurred. Normally –3 would indicate that the PAL procedure itself has failed.

Implication: SAL firmware that assumes self-test errors will be reported with *status* = 0 may not function correctly.

Workaround: When PAL_TEST_PROC returns *status* = –3, SAL should check the *self-test_state* to obtain more information about the self-test error and report the error.

Status: For the steppings affected, see the Summary Table of Changes.

Specification Changes

There are no Specification Changes for this revision of the *Intel® Itanium® 2 Processor Specification Update.*

Specification Clarifications

1. Itanium® 2 processor behavior with CMCI to MCA Promotion enabled

The *Itanium® 2-Based Platform Compatible Processors Firmware Guide* (Ref No 12042), Section 4.7.1.1 describes known behavior on the Itanium 2 processor with regard to CMCI to MCA Promotion and should be further clarified as follows:

- On the Itanium 2 processor, if a concurrent frontside bus (FSB) and L3 cache errors occur, PAL will not recognize the FSB error. The only events that may be ignored are UC accesses that get a HITM or FSB transactions that get a hardfail response. The UC HITM case is a manifestation of illegal memory aliasing attribute (MAA) and should not be common. For hardfail responses, an MCA is generated when the poisoned data is consumed so there is no silent data corruption possibility.

Documentation Change

1. Itanium® 2 processor I_{CTERM} termination agent

The following correction will be made to the next revision of the *Intel® Itanium® 2 Processor at 1.0 GHz and 900 MHz Datasheet* (Doc No 250945).

- In Section 2.3, Table 2-2, "Itanium® 2 Processor Package Specification," footnote c. associated with I_{CTERM} is stated incorrectly:
  - — The original text read:

    Maximum termination voltage current on **two** termination agents.
  - — The correct text should read:

    Maximum termination voltage current on **one** termination agent.

2. Correction of pin/signal information tables

The following corrections will be made in the next revision of the *Itanium® 2-Based Platform Compatible Processors Electrical, Mechanical, and Thermal Specifications* (Ref No 12033) and the *Intel® Itanium® 2 Processor at 1.0 GHz and 900 MHz Datasheet* (Doc No 250945)

- Tables 3-1 and 3-2 incorrectly map the DID[9:0]# and ATTR[3:0]# signals to their corresponding address pin names on the second phase of the address.
  - — Currently the table reads: ATTR[7:0]# on Ab[31:24]# and DID[7:0]# on Ab[23:16]#

Pin Name	Signal Name
A024#	AA24#/ATTR0#
A025#	AA25#/ATTR1#
A026#	AA26#/ATTR2#
A027#	AA27#/ATTR3#
A028#	AA28#/ATTR4#
A029#	AA29#/ATTR5#
A030#	AA30#/ATTR6#
A031#	AA31#/ATTR7#
A032#	AA32#/AB32#
A033#	AA33#/AB33#
A034#	AA34#/AB34#
A035#	AA35#/AB35#

  - — The correct table should read: ATTR[3:0]# on Ab[35:32]# and DID[9:0]# on Ab[25:16]#

Pin Name	Signal Name
A024#	AA24#/DID8#
A025#	AA25#/DID9#
A026#	AA26#/AB26#
A027#	AA27#/xTPRValue0#
A028#	AA28#/xTPRValue1#
A029#	AA29#/xTPRValue2#
A030#	AA30#/xTPRValue3#
A031#	AA31#/xTPRDisable#
A032#	AA32#/ATTR0#
A033#	AA33#/ATTR1#
A034#	AA34#/ATTR2#
A035#	AA35#/ATTR3#